RAPPORT SYSTEMES DISTRIBUES

PROJET **BLOCKCHAIN**

Promotion 2017/2018

Développeurs : Joel Kisala-Kinavuidi, Manal Lamri.

UFR de Mathématiques et Informatique

7 rue René Descartes

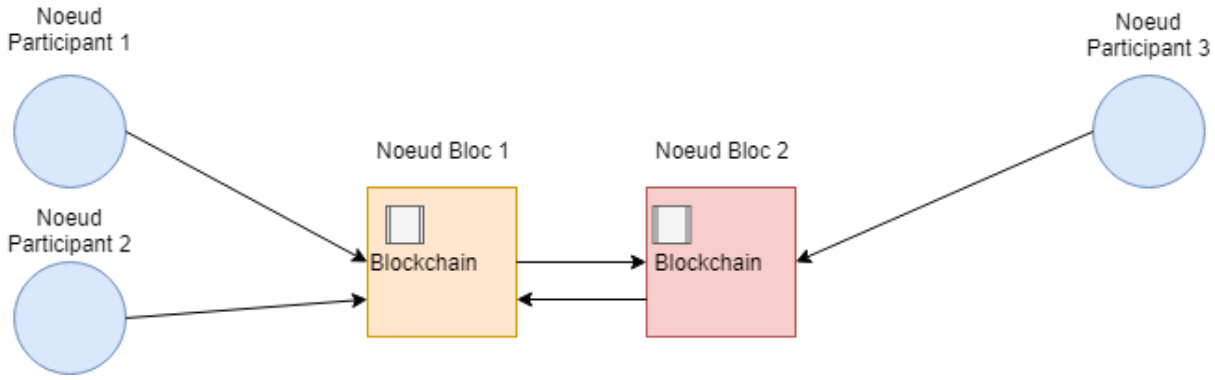
67084 Strasbourg

Conception

Nous avons choisi de développer en utilisant la technologie RMI dans le projet. Elle est entièrement implémentée en Java.

La topologie présentée ici est la suivante : au lancement on crée deux Nœuds Blocs qui tout deux possèdent un objet Blockchain (vides).

Schéma de l'implémentation



L'architecture choisie est une architecture pair à pair pour les Nœuds Blocs. Ces derniers sont à la fois clients et serveurs. Les participants eux, sont seulement clients. Ils se connectent aux Serveurs Nœuds Blocs.

C'est la classe NoeudBlockImpl qui sera partagée aux nœuds blocs clients.

Interface NoeudBlock : celle-ci est implémentée par la classe NoeudBlockImpl dans laquelle les fonctions définies dans l'interface seront implémentées.

Interface Blockchain : celle-ci est implémentée par la classe BlockchainImpl dans laquelle les fonctions définies dans l'interface seront implémentées.

Le programme principal qui lance les Nœuds Blocs se trouve dans la classe Nœud_Bloc. Il est également chargé de créer les différents blocs de la Blockchain.

Un autre programme gère les Nœuds Participants, il se trouve dans la classe Nœud_Participant. Comme expliqué précédemment, cet objet contrairement aux autres a seulement un rôle de client.

Implémentation

Une fois les deux nœuds blocs crées, chacun va tenter de créer des blocs. Le premier qui aura créé la plus longue chaine, obtient le droit de la créer et de la diffuser. Si les deux blocs ont réussi à créer une chaine de même longueur, c'est celui qui a le plus petit Timestamp qui a le droit de diffuser la chaine.

La création d'un bloc se fait comme telle :

Tout d'abord le bloc est sérialisé, c'est-à-dire qu'il est mis sous forme de chaine, puis il est hashé. Le hashage du bloc va sceller le bloc et ne pourra plus être modifié. La chaine sera hashée jusqu'à ce que les 3-4 ou 5 premiers chiffres sont premiers. Si c'est le cas, le bloc est créé.

Les nœuds blocs ont une liste de transaction. Une des transactions possibles est la création d'un bloc.

Tous les nœuds blocs ont ce que l'on appellera une waiting_list. C'est dans cette que sont stockées toutes les opérations en attente. Les nœuds blocs s'échangent leurs waiting_list. Une opération est supprimée de la waiting_list lorsqu'elle a été exécutée. Une opération est exécutée à la création d'un nouveau bloc, on vérifie que les opérations de la waiting_list se retrouvent bien dans le bloc, puis on les supprime de la waiting_list.

Les échanges de données entre nœuds blocs se font de telle sorte que si le Nœud A a besoin d'informations du Nœud B, A va chercher les informations chez B.

Lorsque les participants veulent s'inscrire à un nœud bloc, cette opération est mise dans la waiting_list et n'est exécutée qu'à la création d'un nouveau. Dès que le participant est inscrit, le nœud bloc donne au participant une fraction de sa chaine de blocs.

Si un nœud participant estime qu'il a assez d'argent, il peut choisir de le partager avec un autre participant. Cette opération n'est effective qu'à la création d'un nouveau bloc.

A la création des blocs, si les participants n'ont pas de mérite, ces derniers recevront une fraction égale des blocs créés. Si les participants possèdent un mérite, ils recevront une fraction proportionnelle à ce mérite. Le mérite s'obtient grâce à une compétition entre participants. Si le participant trouve en premier un nombre premier (il lance un random) alors il obtient le mérite (proof_of_work). Quand le nœud bloc créé un bloc, il écoute afin de savoir si des participants ont trouvé un nombre premier.

Jeu de test

Cinq scripts fournis permettent de lancer le programme. Il s'agit des scripts buildN1.sh pour lancer le premier Nœud Block, buildN2.sh qui lance le Nœud Block 2 et buildP1.sh , buildP2.sh, buildP3.sh, buildP4.sh qui lancent respectivement les Nœuds Participants 1, 2, 3 et 4.

Les fichiers doivent être lancés séparément dans des terminaux différents. BuildN1 et buildN2 doivent être exécutés en simultanées (avec une marge d'une dizaine de secondes).

Il existe également un fichier destroy.sh qui va mettre fin à toutes les connexions RMI.

Pour lancer les fichiers il suffit d'écrire en ligne de commande ./<nom fichier>

Exemple : ./buildN1.sh

Difficultés rencontrées

Nous avons eu des difficultés lors de la conception de notre projet. Nous devions définir quelles classes seraient accessibles et si elles seraient clientes ou serveurs, finalement nous avons opté pour l'architecture peer to peer.

Il a été également compliqué au départ de comprendre à quoi avaient concrètement accès les clients, s'ils pouvaient modifier les objets récupérer sur le serveur et si ces modifications seraient prises en compte par le serveur ou si elles seraient contenues que dans la copie qu'en a fait le client.

Nous avons vraiment pris plaisir à développer ce projet mettant en pratique des notions actuelles comme la Blockchain et la cryptomonnaie.